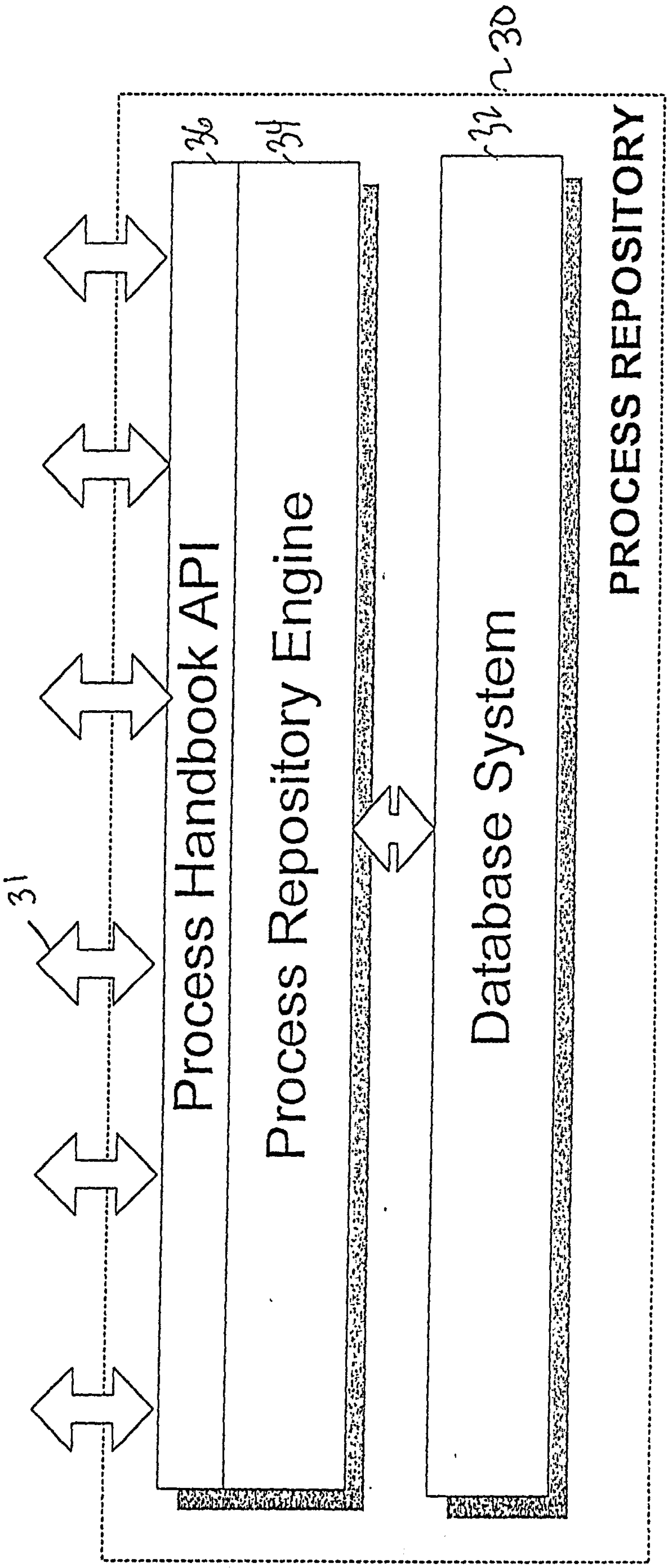

30
PROCESS REPOSITORY
36
34
32
31
Process Handbook API
Process Repository Engine
Database System

FIG. 1

Computer System and Computer Implemented
Process for Representing Software System
Descriptions and for Generating Executable...
Dellarocas, et al.
Docket No.: M0872/7013

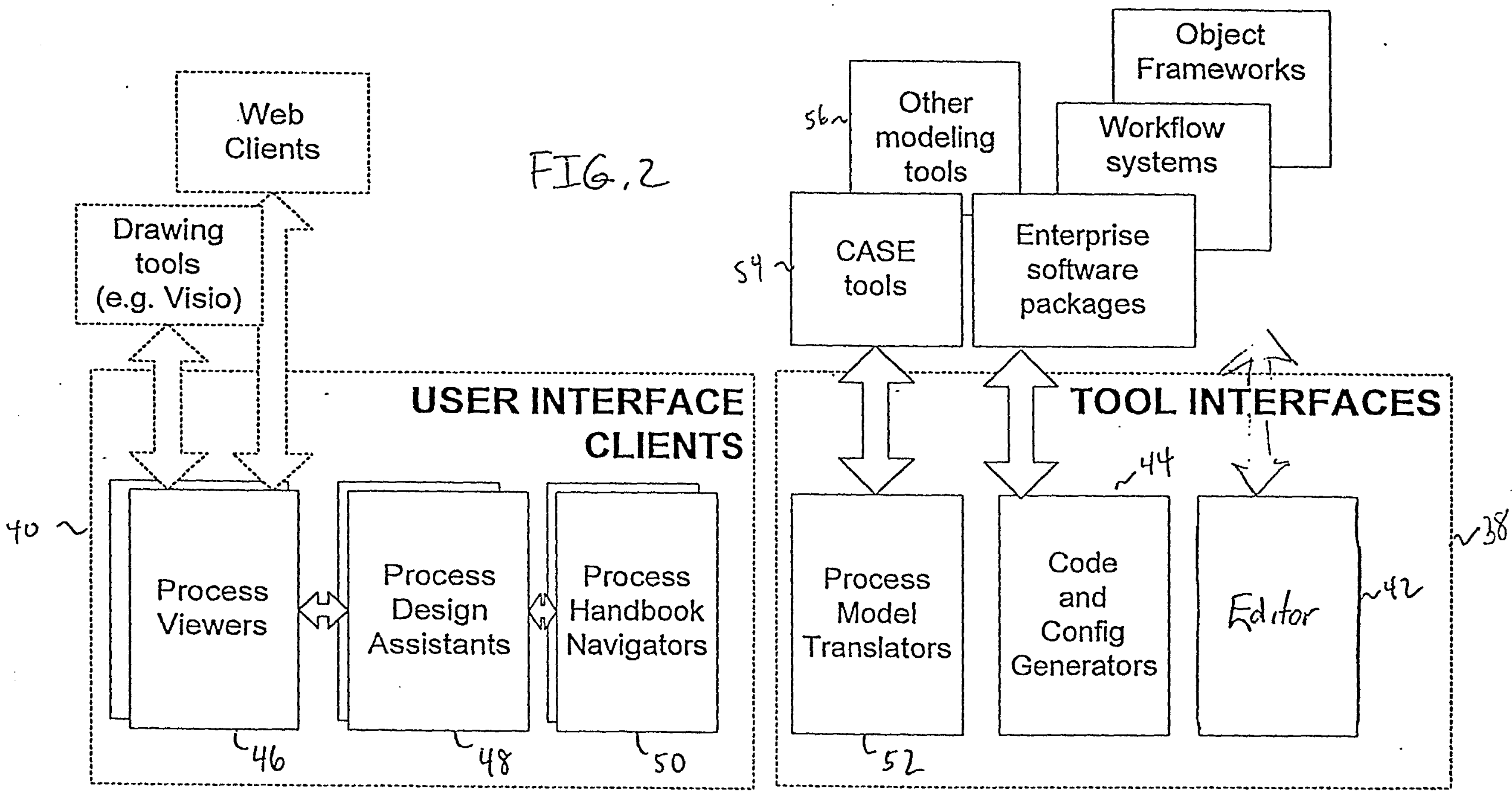


FIG. 2

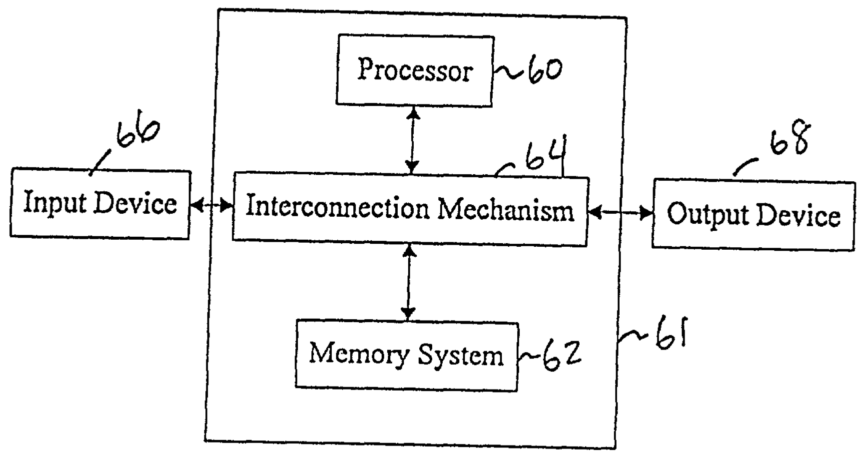

Processor
60
66
64
68
Input Device
Interconnection Mechanism
Output Device
Memory System
62
61

Fig. 3

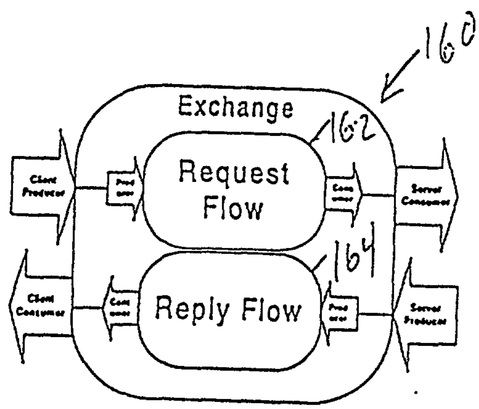

160
Exchange
162
Request Flow
164
Reply Flow
Client Producer
Server Consumer
Client Consumer
Server Producer

FIG. 5

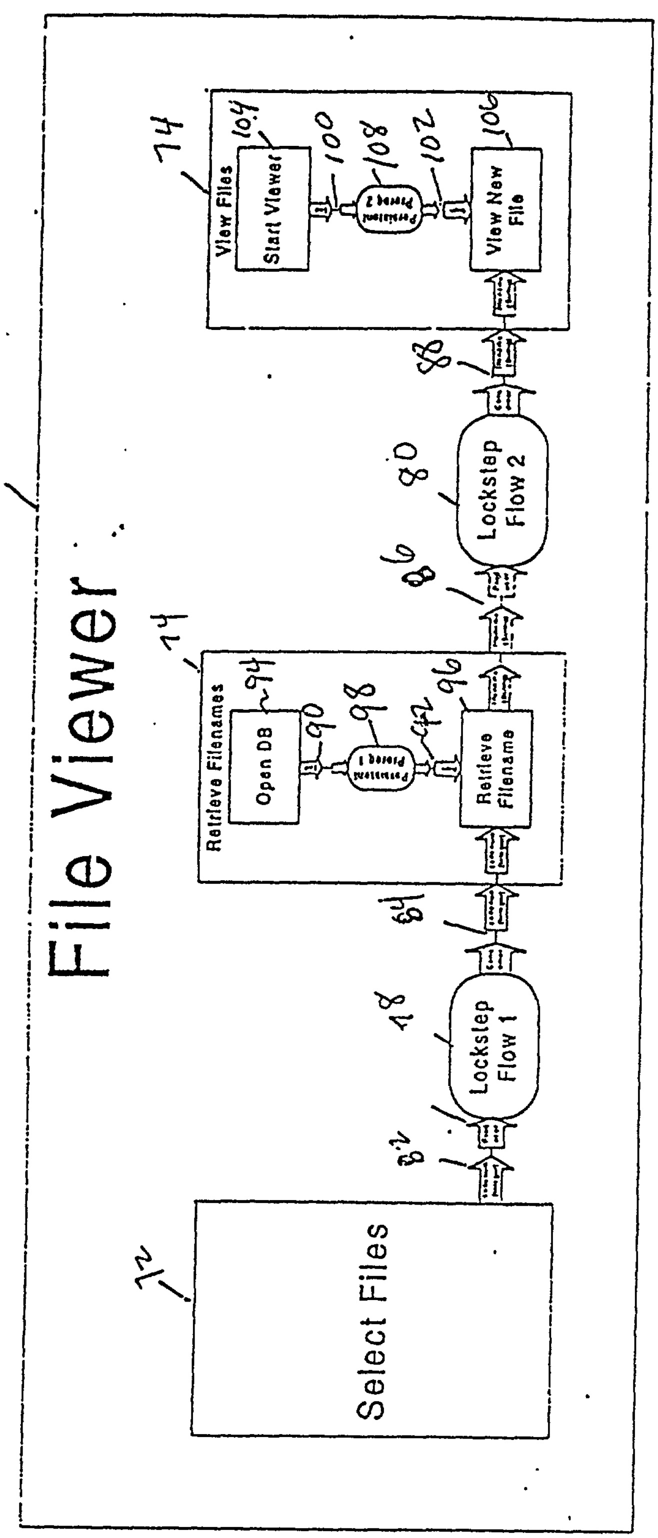

File Viewer
Select Files
Lockstep Flow 1
Retrieve Filenames
Open DB
Retrieve Filename
Lockstep Flow 2
View Files
Start Viewer
View New File
70
72
74
78
80
82
84
86
88
90
94
96
98
100
102
104
106
108

FIG. 4

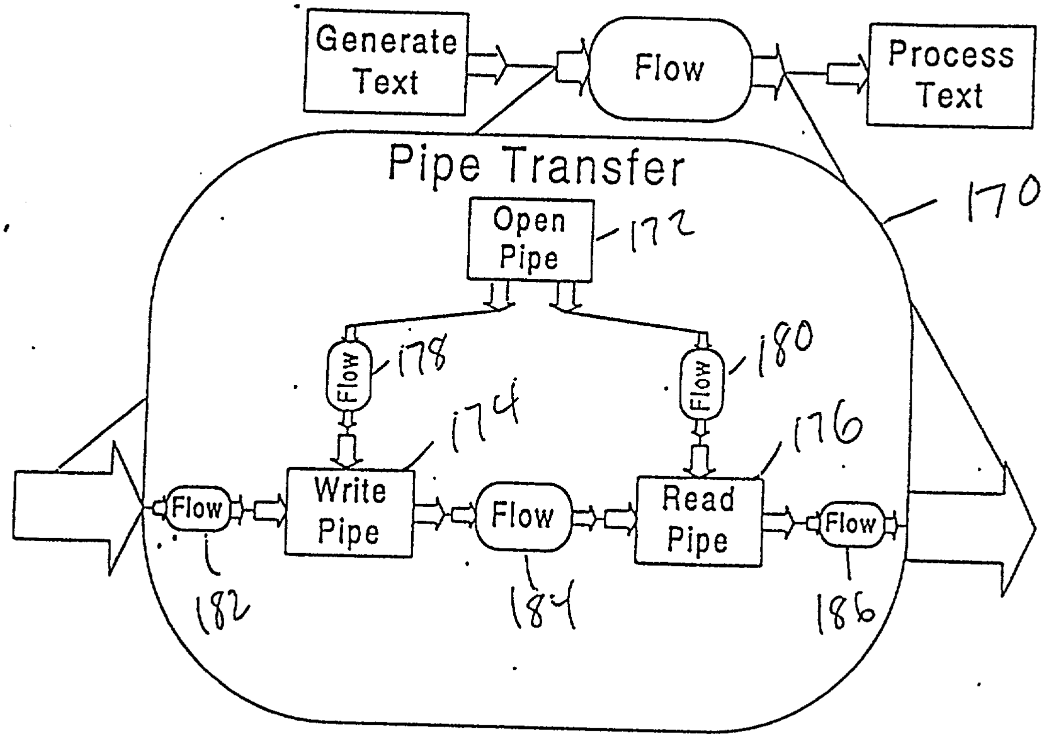


FIG.6

FIG. 7(a)

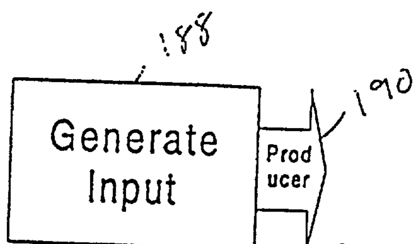


FIG. 7(b)

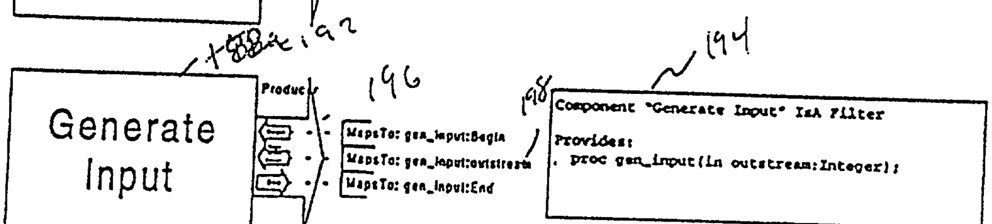
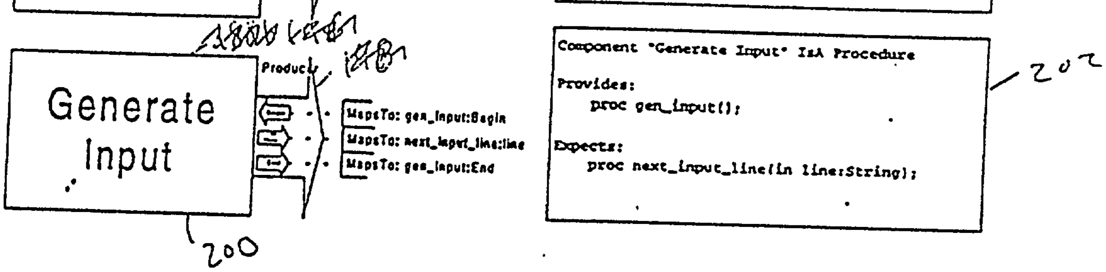


FIG. 7(c)



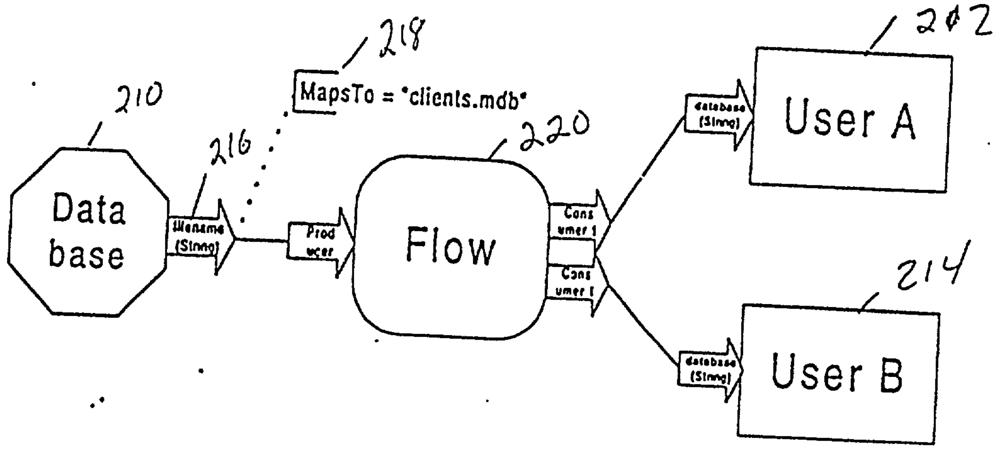

218
MapsTo = "clients.mdb"
210
216
220
202
Data base
Prod ucer
Flow
Cons umer 1
User A
214
User B

FIG. 8

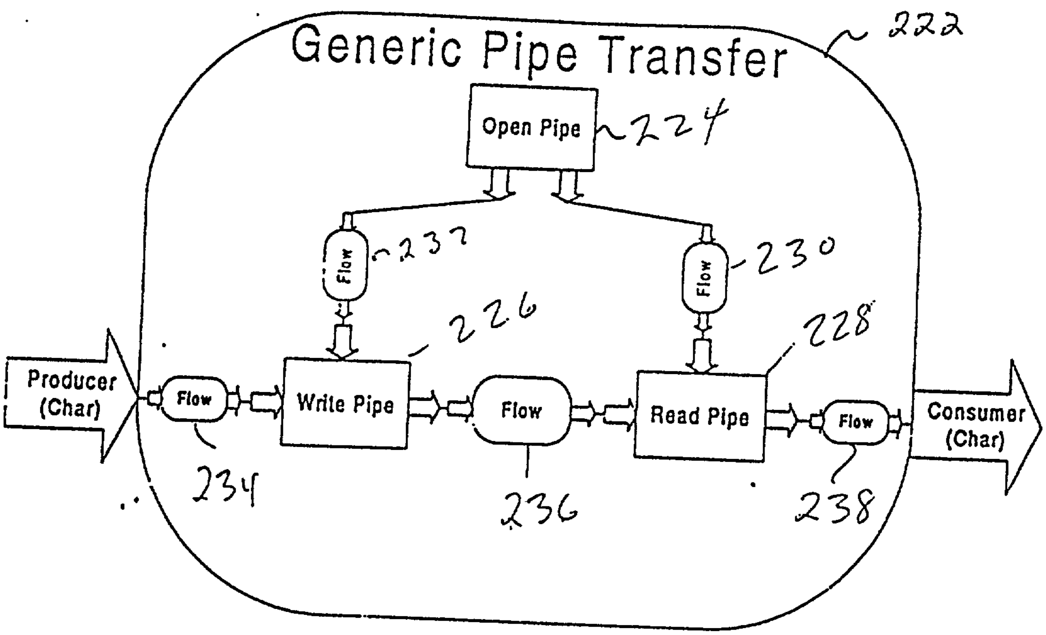

Generic Pipe Transfer
222
Open Pipe
224
Flow
232
Flow
230
226
228
Producer (Char)
Flow
Write Pipe
Flow
Read Pipe
Flow
Consumer (Char)
234
236
238

FIG. 9

PH_Object - 110

Unique Id	112
Name	114
Kind	116
Creator, etc.	118

FIG. 10

Parents -120

Object Id	122
Parent Id	124

FIG. 11

Decomposition - 130

Owner Id	132
Slot Id	134
Slot Value	136
Kind	138
Additional info...	140

FIG. 12

Connectors - 142

Connector Id	144
Owner Id	146
Endpoint 1	148
Endpoint 2	150

FIG. 13

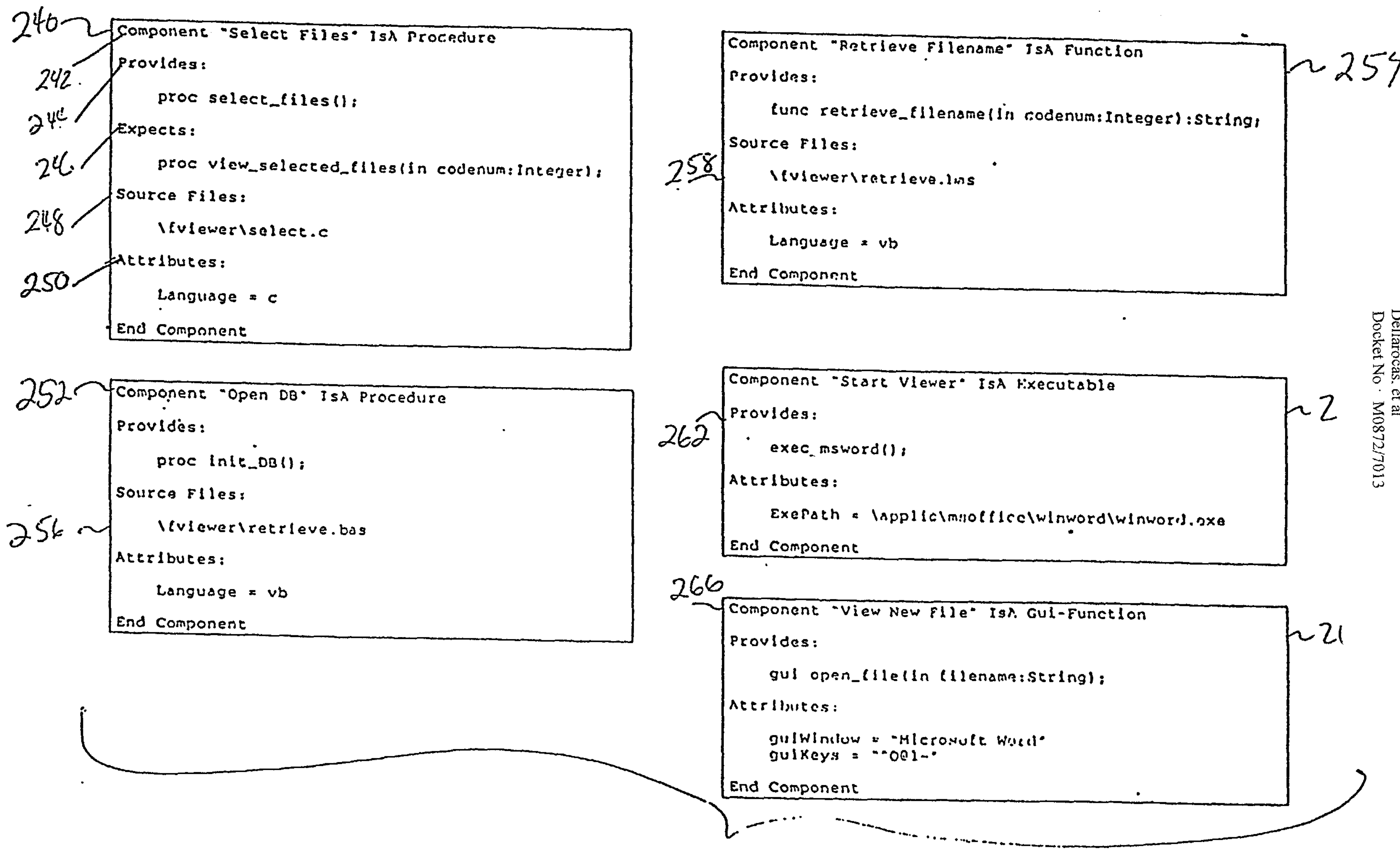

240
242
244
246
248
250
Component "Select Files" IsA Procedure
Provides:
proc select_files();
Expects:
proc view_selected_files(in codenum:Integer);
Source Files:
\fviewer\select.c
Attributes:
Language = c
End Component
252
256
Component "Open DB" IsA Procedure
Provides:
proc init_DB();
Source Files:
\fviewer\retrieve.bas
Attributes:
Language = vb
End Component
254
258
Component "Retrieve Filename" IsA Function
Provides:
func retrieve_filename(in codenum:Integer):String;
Source Files:
\fviewer\retrieve.lws
Attributes:
Language = vb
End Component
2
262
Component "Start Viewer" IsA Executable
Provides:
exec_msword();
Attributes:
ExePath = \applic\msoffice\winword\winword.exe
End Component
21
266
Component "View New File" IsA Gui-Function
Provides:
gui open_file(in filename:String);
Attributes:
guiWindow = "Microsoft Word"
guiKeys = "^O@1~"
End Component

FIG. 14

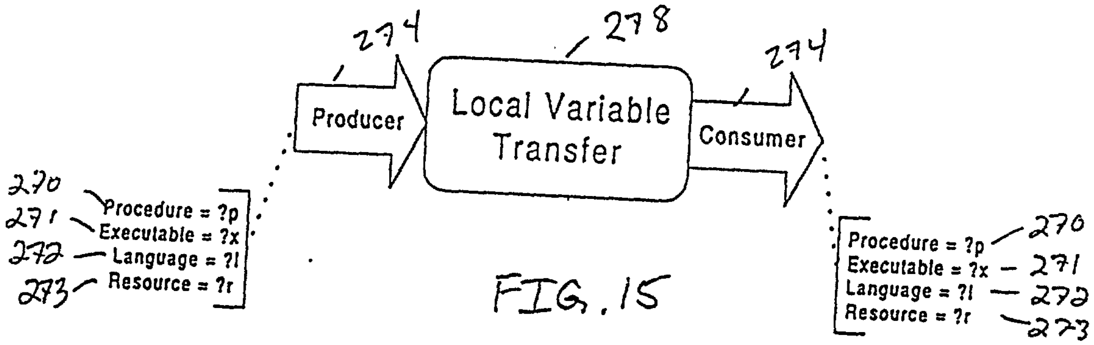


FIG. 15

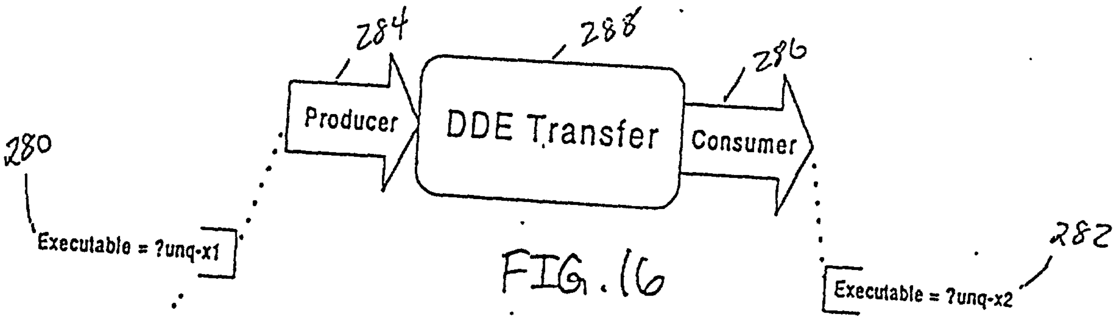


FIG. 16

FIG. 18

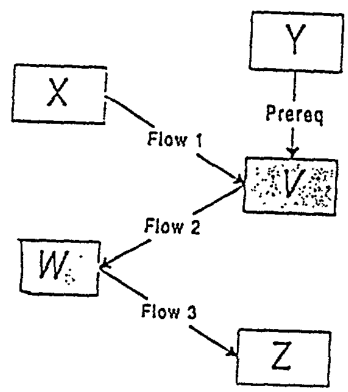
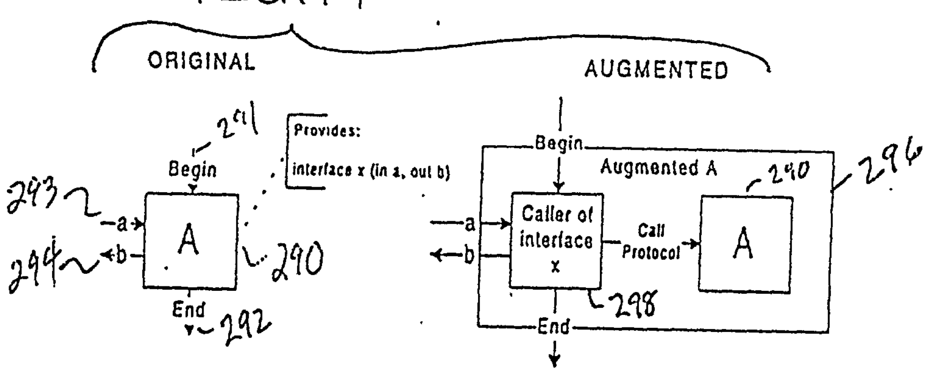


FIG. 19



```
Check_Compatibility( aport, dport )
--                aport = activity port
--                dport = dependency port
-- Returns:       SUCCESS if ports can be legally connected, FAILURE otherwise
-- Uses:          Match_Values( va, vd )

460 - If one port is composite and the other is atomic then return FAILURE.

462 - If both ports are composite then recursively match subports.

464 - If both ports are atomic then
       If aport is same as or a specialization of dport then
               For each attribute defined at both ports (including inherited attributes)
                       If both ends have a value then  call Match_Values
               472     else If one end refers to a variable then
                               If variable has a value then call Match_Values
                               else    Set variable and its equivalence class to value at other end
               470 -                   Return SUCCESS.
                       else If both ends refer to variables
                               If one or both variables have values then do as above.
                               If no variable has a value then
                       474 -           Unify both variables into an equivalence class
                                       Return SUCCESS.
466 -   else return FAILURE.

468:
Match_Values( va, vd )
--                va = value of attribute at activity side
--                vd = value of attribute at dependency side
-- Returns:       SUCCESS if values match, FAILURE otherwise.

 If values are identical then return SUCCESS.
 If values are pointers to language elements then

476 -   If va is a specialization of vd*
                then return SUCCESS.
        *Exception: when comparing resources of consumer ports the opposite specialization relationship
must hold.
 Return FAILURE
```

Fig. 17

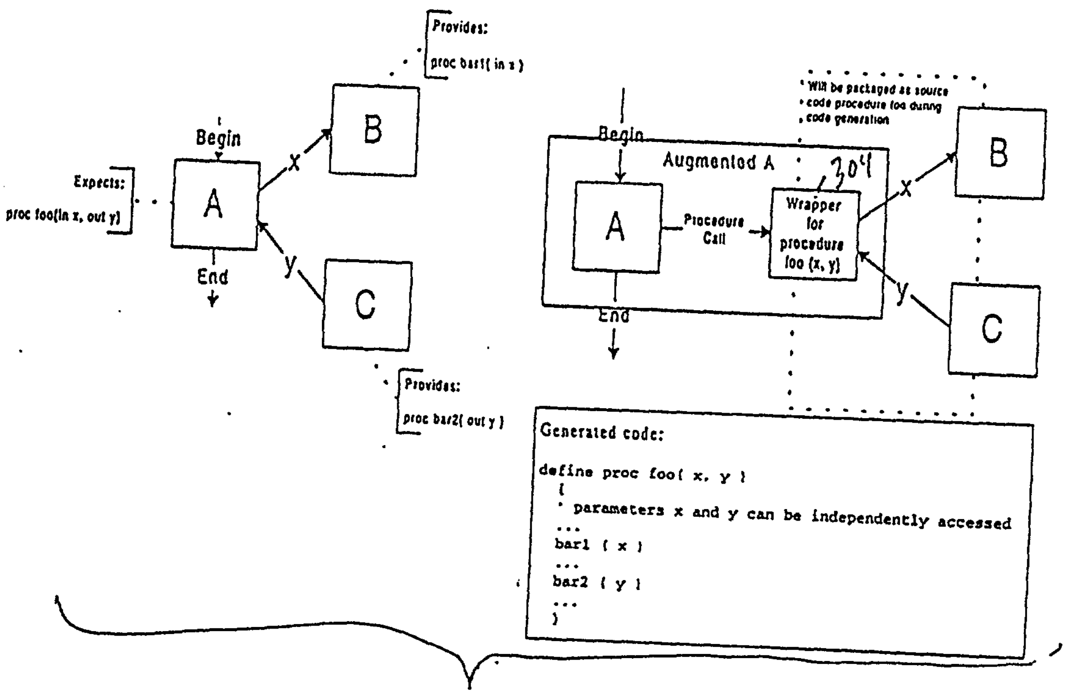

Provides:
proc bar1(in x)
Expects:
proc foo(in x, out y)
Begin
A
B
x
y
C
End
Provides:
proc bar2(out y)
* Will be packaged as source
* code procedure foo during
* code generation
Begin
Augmented A
304
A
Procedure Call
Wrapper for procedure foo (x, y)
B
x
y
C
End
Generated code:
define proc foo(x, y)
{
* parameters x and y can be independently accessed
...
bar1 (x)
...
bar2 (y)
...
}

FIG. 20

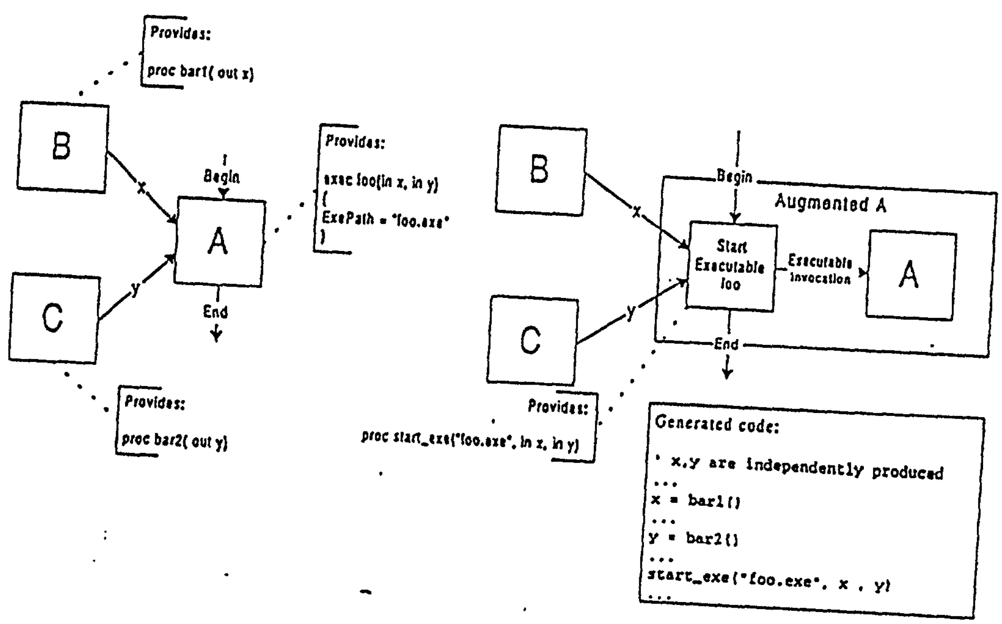

Provides:
proc bar1(out x)
B
x
Begin
A
C
y
End
Provides:
exec foo(in x, in y)
{
ExePath = "foo.exe"
}
Provides:
proc bar2(out y)
B
x
Begin
Augmented A
Start Executable foo
Executable invocation
A
C
y
End
Provides:
proc start_exe("foo.exe", in x, in y)
Generated code:
* x,y are independently produced
...
x = bar1()
...
y = bar2()
...
start_exe("foo.exe", x , y)
...

FIG. 21

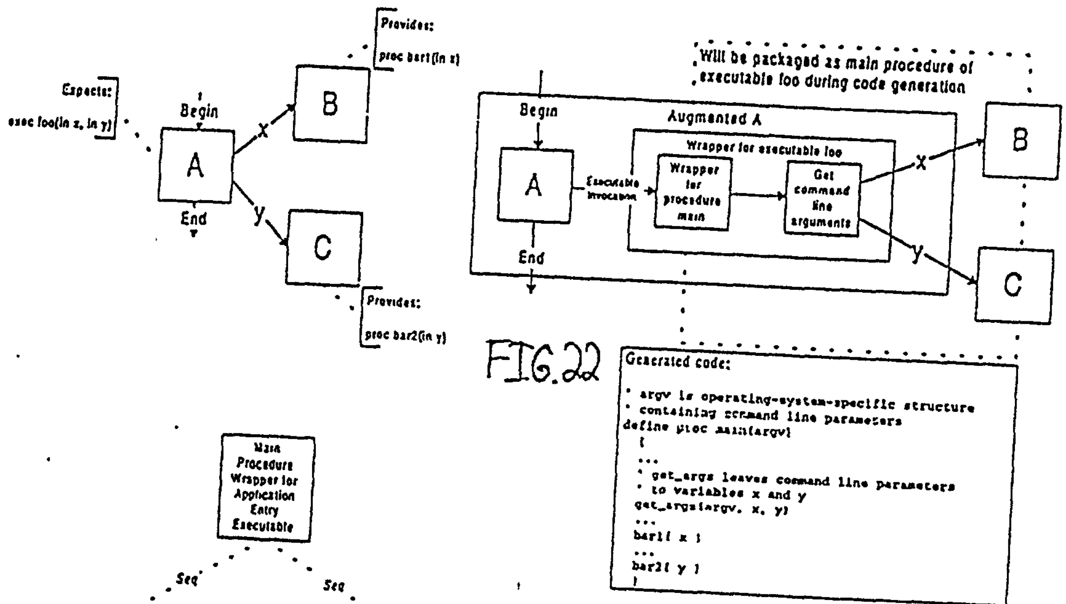


FIG.22

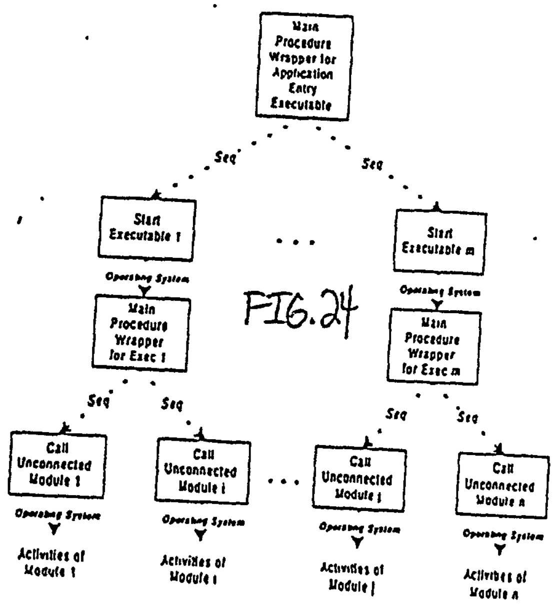


FIG.24

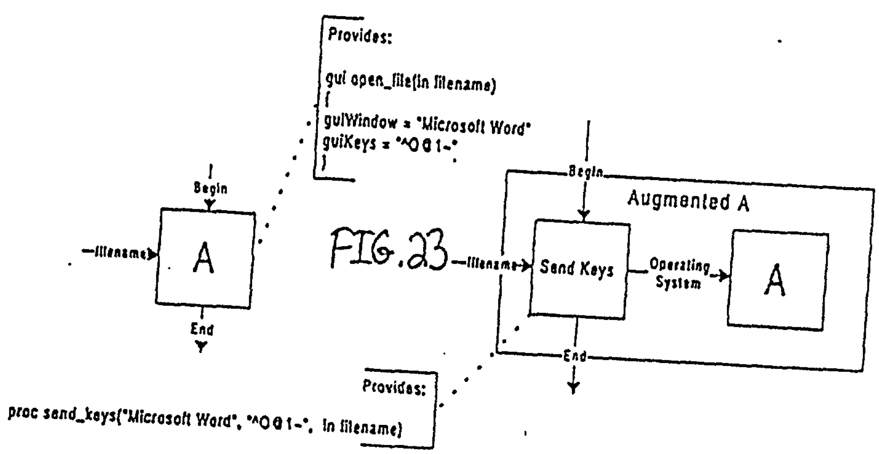


FIG.23

Input: A diagram consisting of activities and dependencies
Output: A set of executable files implementing the target application

330. Decouple interface dependencies
332. Specialize generic design elements
334. Connect all modules to control
336. Generate executable code

Fig. 25

338 - Recursively scan all activities in the application graph.
For every activity associated with a code-level component,
340 - Scan all provided and expected interface definitions of the associated component.
For every provided interface,
342 - Get the interface kind.
If a caller activity has been defined for that interface kind,
344 - Check for "perfect match" special cases
If no "perfect match" interface is found at the other end,
346 - Replace the original primitive activity with a composite pattern that includes a caller activity.
For every expected interface,
348 - Get the interface kind.
If a wrapper activity has been defined for that interface kind,
350 - Check for "perfect match" special cases
If no "perfect match" interface is found at the other end,
352 - Replace the original primitive activity with a composite pattern that contains a wrapper activity.

Fig. 26

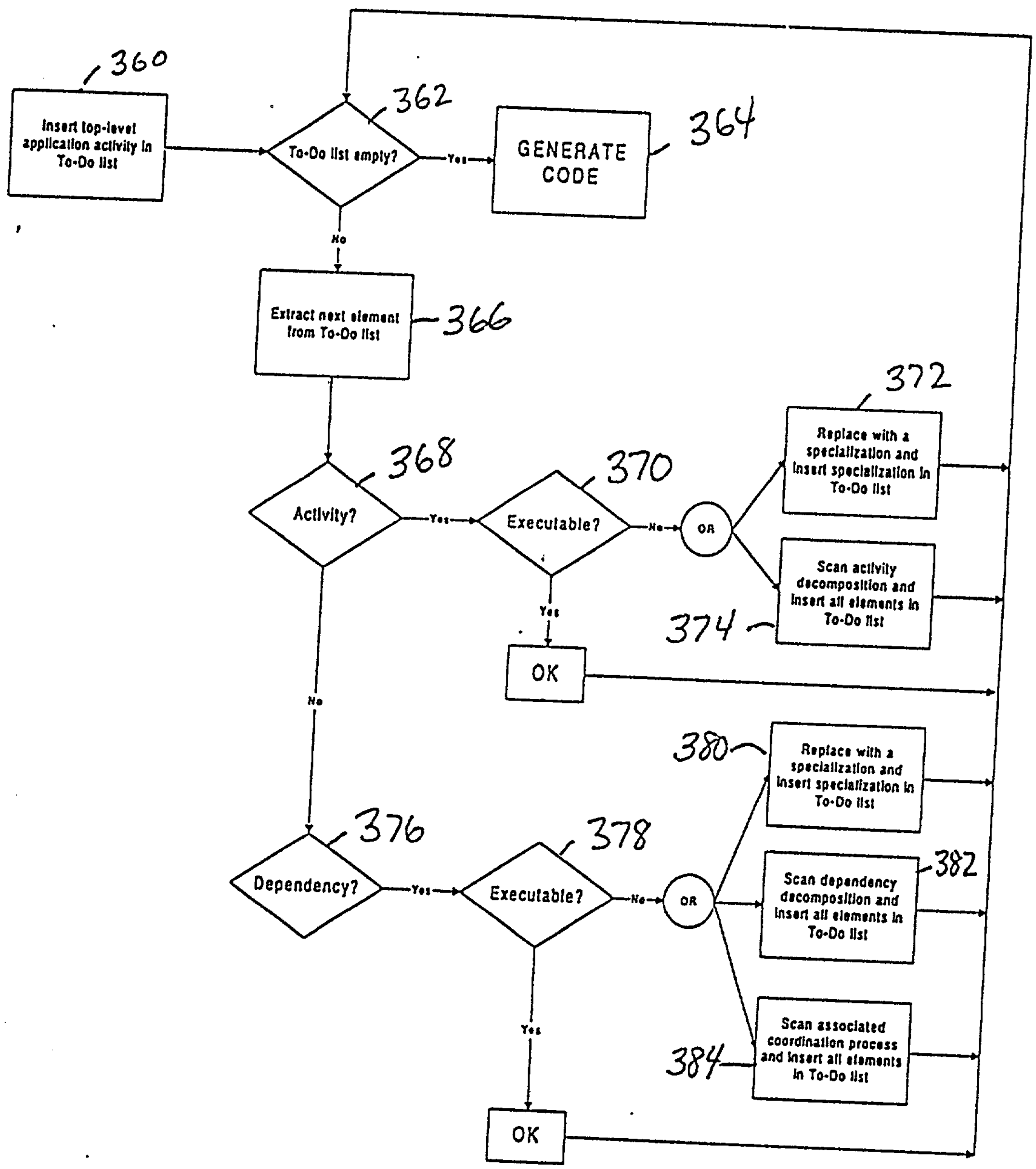


FIG. 27

10101" 09520891

390: Scan graph and build a *to-do list* containing,
- all generic atomic activities (i.e. atomic activities not associated with a code-level component)
- all unmanaged dependencies

Repeat the following two operations until to-do list becomes empty,

392:a. Extract the next generic atomic activity.
For each executable specialization of that activity stored in the design repository,
396 Apply the compatibility checking algorithm
398 If at least two matching executable specializations are found,
Ask user to select between them.
Otherwise,
Repeat while user input is invalid:
400 Ask user to provide a specialization for the activity.
Check validity of user-supplied activity (if must pass the compatibility checking algorithm and either be *atomic and executable* or *composite*
402 Permanently store new activity in the repository.
Replace generic activity with selected or user-supplied specialization.
406 Apply Stage 1 of the algorithm to the replacing activity.
If replacing activity is composite and generic,
408 Scan activity decomposition and add all generic atomic activities and unmanaged dependencies found to the to-do list

394:b. Extract the next unmanaged dependency
For each coordination process* associated with a specialization of that dependency stored in the design repository,
410 Apply the compatibility checking algorithm.
If at least two matching coordination processes are found,
412 Ask the user to select among them.
Otherwise,
Repeat while user input is invalid:
414 Ask user to provide a compatible coordination process.
416 Check validity of user-supplied process (it must pass the compatibility checking algorithm and either be *atomic and executable* or *composite*)
418 Permanently store new process in the repository.
420 Manage dependency with the selected or user-supplied coordination process.
422 Apply Stage 1 (Fig. 26) of the algorithm to the managing coordination process.
If managing coordination process is composite,
424 Scan process decomposition and add all generic atomic activities and unmanaged dependencies found to the to-do list.

* The term coordination process here also includes atomic software connectors associated with executable dependencies.

Fig. 28

430. Scan the application graph and find all source modules that are not connected to a source of control.

432. Introduce a set of *packaging* executable components, one per host machine and per language for which unconnected source modules exist.

434. Package calls to unconnected source modules inside the main program of the packaging executable corresponding to the host machine and language of each module.

436. Scan the application graph and find all executable programs that are not connected to a source of control.

438. Introduce an *application entry* executable component into the system.

440. Package invocation statements for all unconnected executables inside the main program of the application entry component.

Fig. 29

442 Scan graph and divide into *sequential block subgraphs**.

For each subgraph,

444 Topologically order activities according to their sequentialization interdependencies.

446 Generate a call statement for each activity.

448 Generate a local variable declaration for each local variable coordination process.

450 Generate appropriate headers and footers for the enclosing sequential block.

452 Save resulting sequential block code into a file.

For each target executable:

454 Collect all source and object files of the executable**.

456 Compile files and place resulting executable into target application directory.

* Sets of activiteis that will be packaged in the same sequential code block.

**The files are (1) all source and object files referenced in the component descriptions of all activities to be included in the executable, and (2) all coordination code source files generated by step 442 for the target executable.

Fig. 30

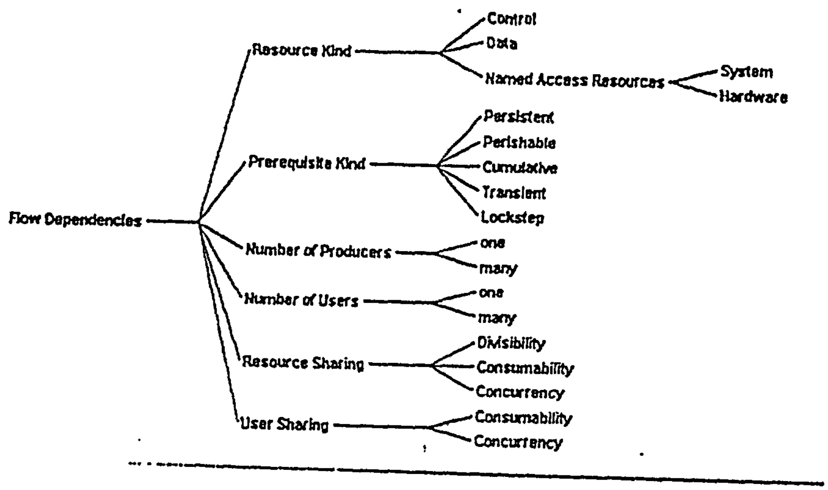

Flow Dependencies
Resource Kind
Control
Data
Named Access Resources
System
Hardware
Prerequisite Kind
Persistent
Perishable
Cumulative
Transient
Lockstep
Number of Producers
one
many
Number of Users
one
many
Resource Sharing
Divisibility
Consumability
Concurrency
User Sharing
Consumability
Concurrency

FIG. 31

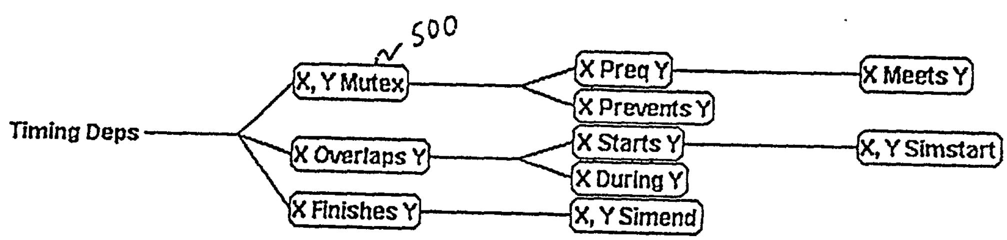

500
Timing Deps
X, Y Mutex
X Preq Y
X Meets Y
X Prevents Y
X Overlaps Y
X Starts Y
X, Y Simstart
X During Y
X Finishes Y
X, Y Simend

FIG. 32